下载APP

07 | 海量数据处理技术回顾：为什么分布式会遇到 CAP 难题？

2022-03-02 李智慧

《李智慧 · 高并发架构实战课》 课程介绍 >

讲述：李智慧
时长 16:24 大小 15.02M

你好，我是李智慧。

在这个模块的几个案例中，我们都需要处理海量的数据，需要用到海量的存储介质，其实海量数据本质上就是一种磁盘资源敏感的高并发场景。

我们说过，为了应对资源不足的问题，我们常采用水平伸缩，即分布式的方案。数据存储的分布式问题是所有分布式技术中最具挑战性的，因为相对于“无状态”（stateless）的计算逻辑（可执行程序），数据存储是“**有状态**”（stateful）的。无状态的计算逻辑可以在任何一台服务器执行而结果不会改变，但有状态的数据却意味着数据存储和计算资源绑定：每一个数据都需要存储在特定的服务器上，如果再增加一台空的服务器，它没有数据，也就无法提供数据访问，无法实现伸缩。

数据存储的“有状态”特性还会带来其他问题：为了保证数据存储的可靠性，数据必须多备份存储，也就是说，同一个数据需要存储在多台服务器上。那么又如何保证多个备份的数据是一致的？

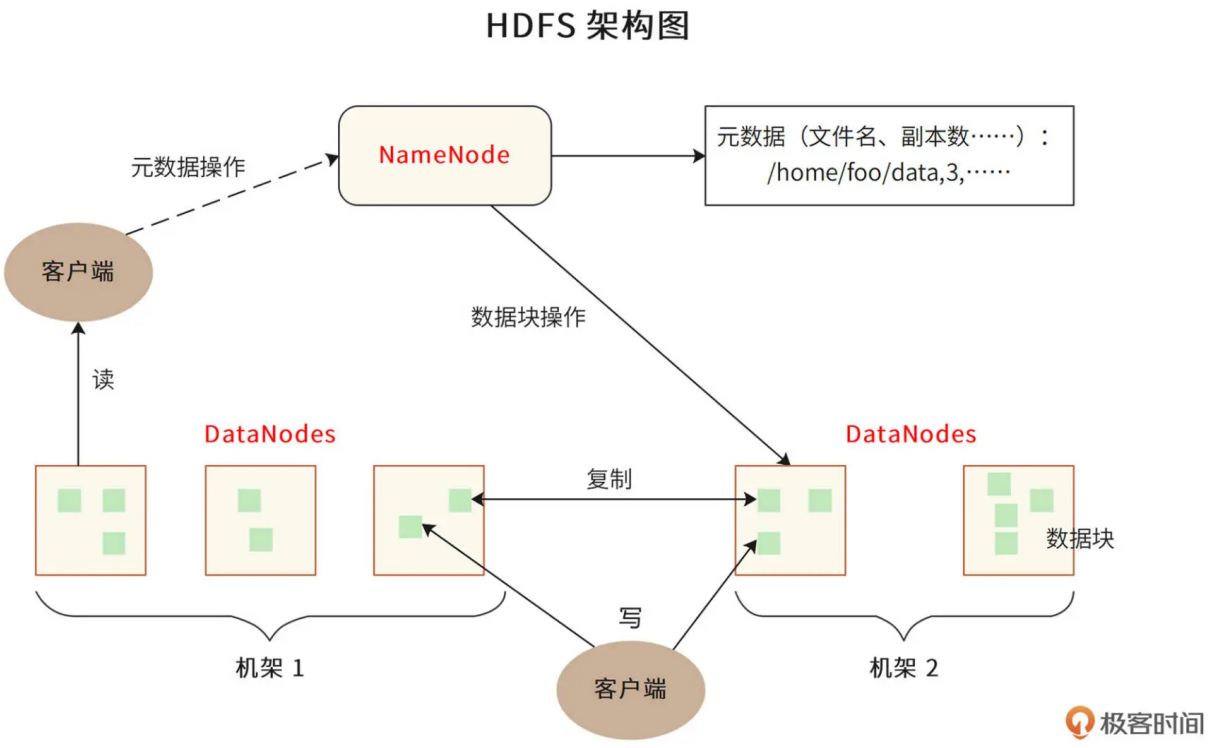
因此，海量数据存储的核心问题包括：如何利用分布式服务器集群实现海量数据的统一存储？如何正确选择服务器写入并读取数据？为了保证数据的高可用性，如何实现数据的多备份存储？数据多备份存储的时候，又如何保证数据的一致性？

为了解决这些问题，在这个模块的案例设计中，我们使用了多个典型的分布式存储技术方案：分布式文件系统 HDFS、分布式 NoSQL 数据库 HBase、分布式关系数据库。下面我们就来回顾这几个典型技术方案。你可以再重新审视一下，我们案例中的技术选型是否恰当，是否有改进的空间。

HDFS

这个模块中，我们用 HDFS 作为短 URL、爬虫下载文件、短视频文件的存储方案。

HDFS，即 Hadoop 分布式文件系统，其架构如下。



HDFS 的关键组件有两个，一个是 NameNode，另一个是 DataNode。

NameNode 负责整个分布式文件系统的元数据管理，也就是文件路径名、访问权限、数据块 ID、存储位置等信息。而 DataNode 负责文件数据的存储和读写操作，HDFS 将文件数据分割成若干数据块（Block），每个 DataNode 存储一部分数据块，这样文件就分布存储在了整个 HDFS 服务器集群中。

HDFS 集群会有很多台 DataNode 服务器（一般几百到几千不等），每台服务器配有数块硬盘，整个集群的存储容量大概在几 PB 到数百 PB。通过这种方式，HDFS 可以存储海量的文件数据。

HDFS 为了保证数据的高可用，会将一个数据块复制为多份（缺省情况为 3 份），并将多份相同的数据块存储在不同的服务器上，甚至不同的机架上。这样当有硬盘损坏，或者某个 DataNode 服务器宕机，甚至某个交换机宕机，导致其存储的数据块不能访问的时候，客户端会查找其备份的数据块进行访问。

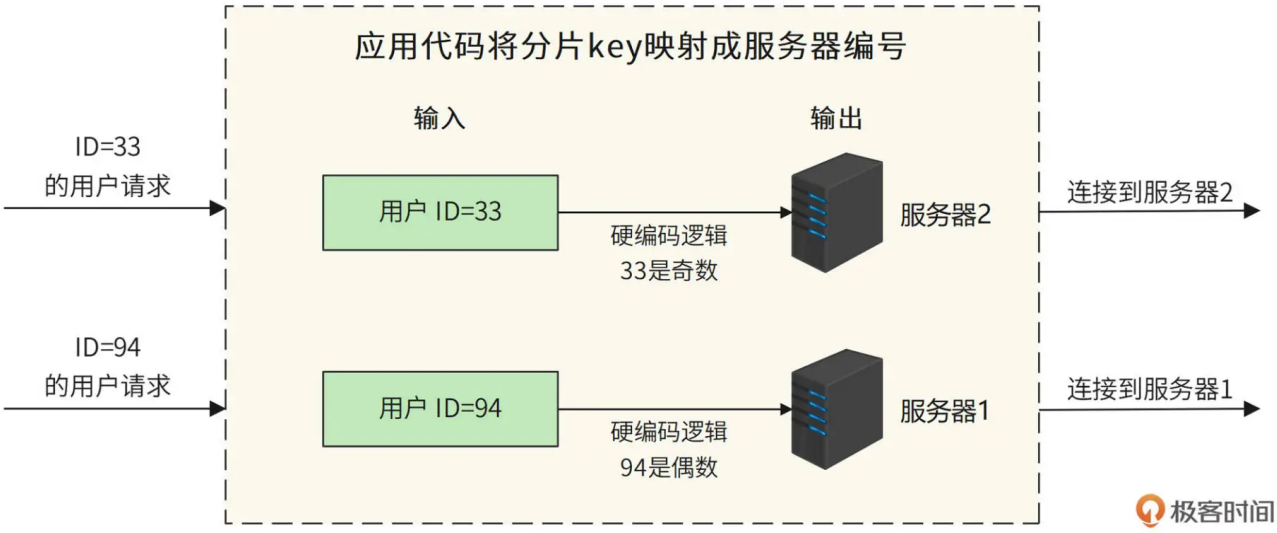
HDFS 的典型应用场景是大数据计算，即使用 MapReduce、Spark 这样的计算框架来计算存储在 HDFS 上的数据。但是作为一个经典的分布式文件系统，我们也可以把 HDFS 用于海量文件数据的存储与访问，就像我们在这个模块的案例中那样。

分布式关系数据库

我们在网盘案例中，使用了分片的关系数据来存储元数据信息。这是因为关系数据存在存储结构的限制（使用 B+ 树存储表数据），通常一张表的存储上限是几千万条记录。而在网盘的场景中，元数据在百亿以上，所以我们需要将数据分片存储。

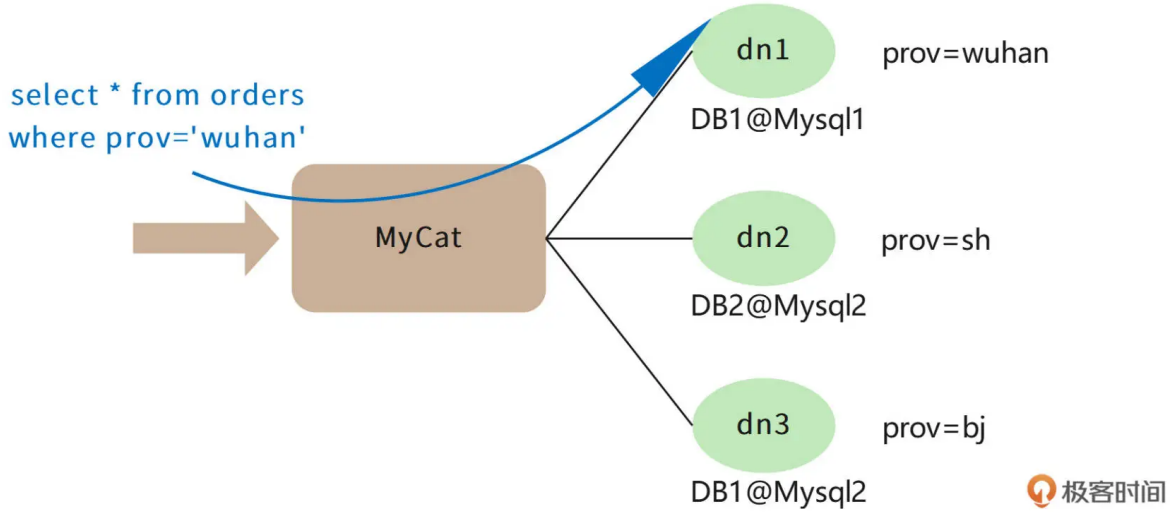
分片的关系数据库，也被称为分布式关系数据库。也就是说，将一张表的数据分成若干片，其中每一片都包含了数据表中一部分的行记录，然后将每一片存储在不同的服务器上，这样一张表就存储在多台服务器上了。通过这种方式，每张表的记录数上限可以突破千万，保存百亿甚至更多的记录。

最简单的数据库分片存储可以采用硬编码的方式，我们在程序代码中直接指定把一条数据库记录存放在哪个服务器上。比如像下图这样，要将用户表分成两片，存储在两台服务器上，那么我们就可以在程序代码中根据用户 ID 进行分片计算，把 ID 为偶数（如 94）的用户记录存储到服务器 1，ID 为奇数（如 33）的用户记录存储到服务器 2。



但是硬编码方式的缺点比较明显。如果要增加服务器，那么就必须修改分片逻辑代码，这样程序代码就会因为非业务需求产生不必要的变更；其次，分片逻辑会耦合在业务逻辑的程序代码中，修改分片逻辑或业务逻辑，都可能影响另一部分代码，从而出现 Bug。

我们可以使用分布式关系数据库中间件来解决这个问题，在中间件中完成数据的分片逻辑，这样对应用程序是透明的。我们常用的分布式关系数据库中间件是 MyCAT，原理如下图。



MyCAT 是针对 MySQL 数据库设计的，应用程序可以像使用 MySQL 数据库一样连接 MYCAT，提交 SQL 命令。MyCAT 在收到 SQL 命令以后，查找配置的分片逻辑规则。

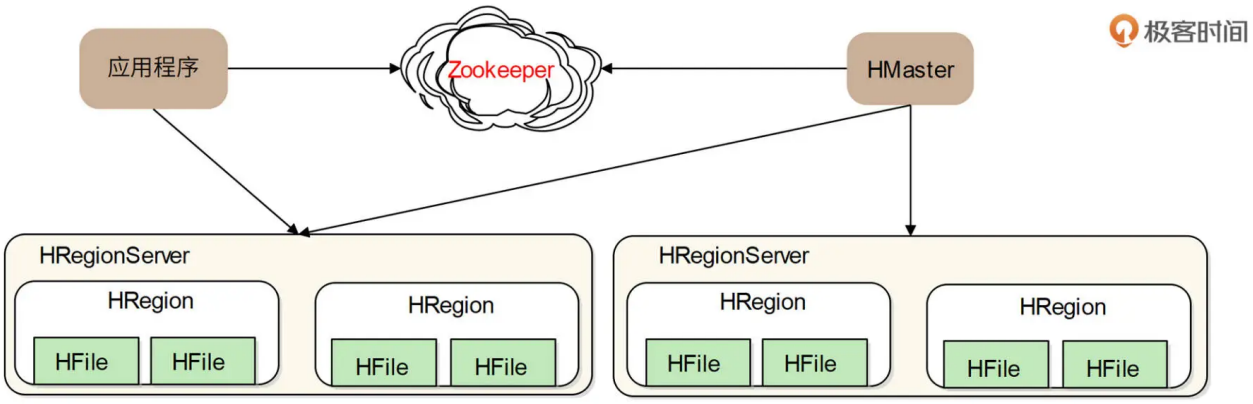
比如上图中，我根据地区进行数据分片，把不同地区的订单存储在不同的数据库服务器上。那么 MyCAT 就可以解析出 SQL 中的地区字段 prov，根据这个字段连接相对应的数

据库服务器。例子中 SQL 的地区字段是“wuhan”，而在 MyCAT 中配置“wuhan”对应的数据库服务器是 dn1，所以用户提交的这条 SQL 最终会被发送给 DB1@Mysql1 数据库进行处理。

HBase

分布式关系数据库可以解决海量数据的存储与访问，但是关系数据库本身并不是分布式的，需要通过中间件或者硬编码的方式进行分片，这样对开发和运维并不友好，于是人们又设计出了一系列天然就是分布式的数据存储系统。因为这些数据存储系统通常不支持关系数据库的 SQL 语法，所以它们也被称为 NoSQL 数据库。

HBase 就是 NoSQL 数据库中较为知名的一个产品。我们的短 URL 数据存储、短视频缩略图存储都使用了 HBase 作为存储方案。上面网盘元数据存储方案使用了分布式关系数据库，事实上，使用 HBase 这样的 NoSQL 数据库会是更好的方案。HBase 架构如下。



HRegion 是 HBase 中负责数据存储的主要进程，应用程序对数据的读写操作都是通过和 HRetion 通信完成的。也就是说，应用程序如果想要访问一个数据，必须先找到 HRegion，然后将数据读写操作提交给 HRegion，而 HRegion 最终将数据存储到 HDFS 文件系统中。由于 HDFS 是分布式、高可用的，所以 HBase 的数据存储天然是分布式、高可用的。

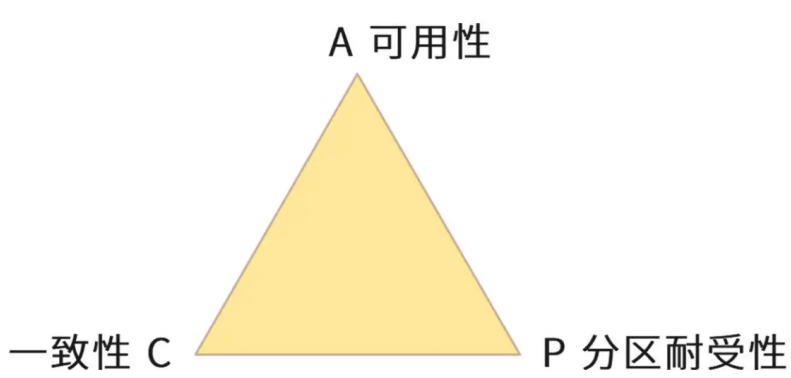
因此 HBase 的设计重点就是 HRegion 的分布式。HRegionServer 是物理服务器，这些服务器构成一个分布式集群，每个 HRegionServer 上可以启动多个 HRegion 实例。当一个 HRegion 中写入的数据太多，达到配置的阈值时，一个 HRegion 会分裂成两个 HRegion，并将 HRegion 在整个集群中进行迁移，以使 HRegionServer 的负载均衡，进而实现 HRegion 的分布式。

应用程序如果想查找数据记录，需要使用数据的 key。每个 HRegion 中存储一段 Key 值区间[key1, key2) 的数据，而所有 HRegion 的信息，包括存储的 Key 值区间、所在 HRegionServer 地址、访问端口号等，都记录在 HMaster 服务器上。因此，应用程序要先访问 HMaster 服务器，得到数据 key 所在的 HRegion 信息，再访问对应的 HRegion 获取数据。为了保证 HMaster 的高可用，HBase 会启动多个 HMaster，并通过 ZooKeeper 选举出一个主服务器。

ZooKeeper

我们在上面提到，分布式数据存储为了保证高可用，需要对数据进行多备份存储，但是多份数据之间可能无法保证数据的一致性，这就是著名的 CAP 原理。

CAP 原理认为，一个提供数据服务的分布式系统无法同时满足数据**一致性**（Consistency）、**可用性**（Availibility）、**分区耐受性**（Patition Tolerance）这三个条件，如下图所示。



其中，一致性的意思是，每次读取数据，要么读取到最近写入的数据，要么返回一个错误，而不是过期数据，这样就能保证数据一致。

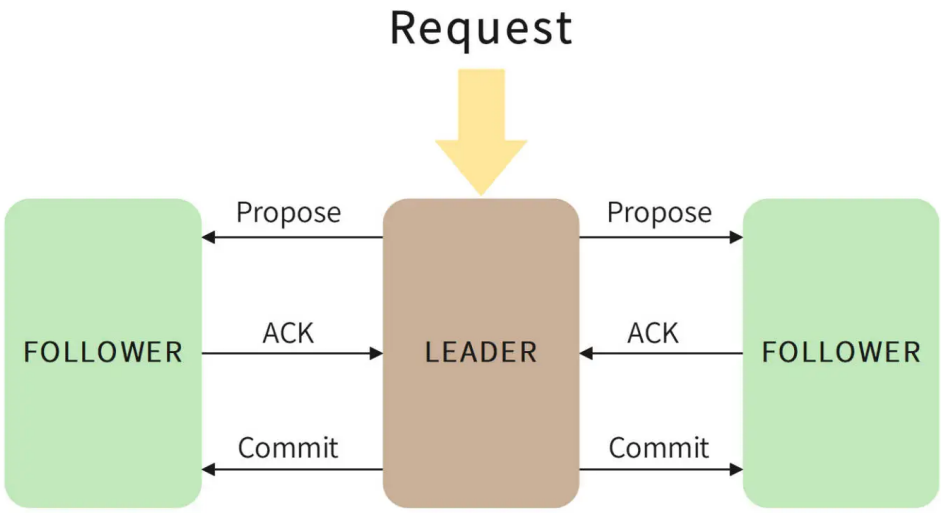
可用性的意思是，每次请求都应该得到一个响应，而不是返回一个错误或者失去响应，不过这个响应不需要保证数据是最近写入的。也就是说，系统需要一直都能正常使用，不会引起调用者的异常，但是并不保证响应的数据是最新的。

分区耐受性的意思是，即使因为网络原因，部分服务器节点之间消息丢失或者延迟了，系统依然应该是可以操作的。

当网络分区失效发生时，要么我们取消操作，保证数据一致性，但是系统却不可用；要么我们继续写入数据，但是数据的一致性就得不到保证。

对于一个分布式系统而言，网络失效一定会发生，也就是说，分区耐受性是必须要保证的，那么可用性和一致性就只能二选一，这就是 CAP 原理。

由于互联网对高可用的追求，大多数分布式存储系统选择可用性，而放松对一致性的要求。而 ZooKeeper 则是一个保证数据一致性的分布式系统，它主要通过一个 ZAB 算法（Zookeeper Atomic Broadcast， Zookeeper 原子广播）实现数据一致性，算法过程如下。



ZooKeeper 集群由多台服务器组成，为了保证多台服务器上存储的数据是一致的，ZAB 需要在这些服务器中选举一个 Leader，所有的写请求都必须提交给 Leader。Leader 服务器会向其他服务器（Follower）发起 Propose，通知所有服务器：“我们要完成一个写操作请求，请大家检查自己的数据状态是否有问题。”

如果所有 Follower 服务器都回复 Leader 服务器 ACK，即没有问题，那么 Leader 服务器会向所有 Follower 发送 Commit 命令，要求所有服务器完成写操作。这样包括 Leader 服务器在内的所有 ZooKeeper 集群服务器的数据，就都更新并保持一致了。如果有两个客户端程序同时请求修改同一个数据，因为必须要经过 Leader 的审核，而 Leader 只接受其中一个请求，数据也会保持一致。

在实际应用中，客户端程序可以连接任意一个 Follower，进行数据读写操作。如果是写操作，那么这个请求会被 Follower 发送给 Leader，进行如上所述的处理；如果是读操作，

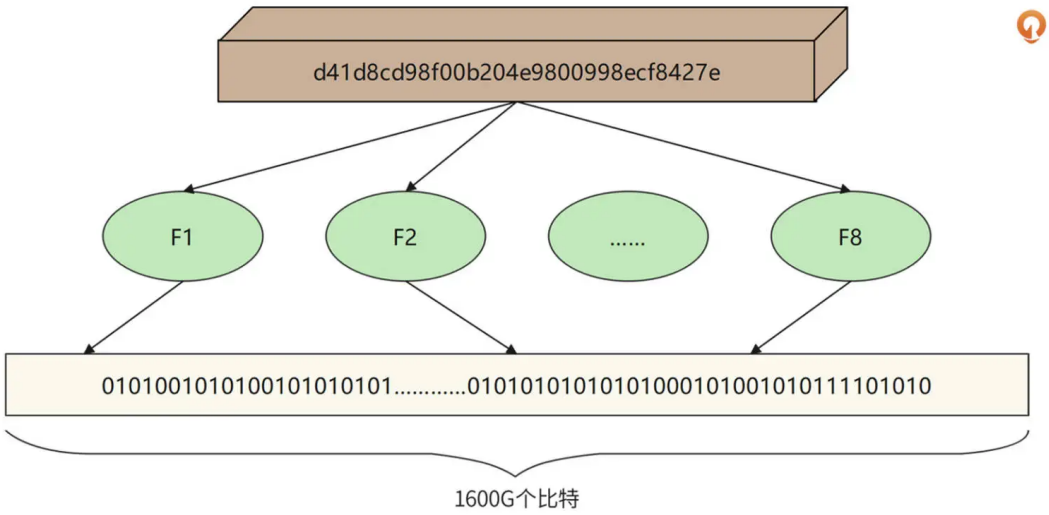
因为所有服务器的数据都是一致的，那么这个 Follower 直接把自己本地的数据返回给客户端就可以了。

因为 ZooKeeper 具有这样的特性，所以很多分布式系统都使用 ZooKeeper 选择主服务器。为了保证系统高可用，像 HDFS 中的 NameNode，或者 HBase 中的 HMaste 都需要主主热备，也就是多台服务器充当主服务器，这样任何一台主服务器宕机，都不会影响系统的可用性。

但是在运行期，只能有一台主服务器提供服务，否则系统就不知道该接受哪台服务器的指令，即出现所谓的系统脑裂，因此系统需要选举主服务器。而 ZooKeeper 的数据一致性特点可以保证只有一台服务器选举成功。在专栏后面的网约车架构案例中，我们也使用了 ZooKeeper 进行服务器管理。

布隆过滤器

我们在短 URL 生成以及网络爬虫的案例中，还使用了布隆过滤器检查内容是否重复，即检查短 URL 或者网页内容的 MD5 是否已经存在。如果用 Hash 表检查重复，千亿级的网页内容 MD5 就需要一个非常大的 Hash 表，内存资源消耗非常大。而用布隆过滤器，使用较小的内存就可以检查海量数据中一个数据是否存在。文件 MD5 重复性检查的布隆过滤器原理如下。



布隆过滤器首先开辟一块巨大的连续内存空间，比如开辟一个 1600G 比特的连续内存空间，也就是 200GB 大的一个内存空间，并将这个空间所有比特位都设置为 0。然后对每个 MD5 使用多种 Hash 算法，比如使用 8 种 Hash 算法，分别计算 8 个 Hash 值，并保证每个 Hash 值是落在这个 1600G 的空间里的，也就是，每个 Hash 值对应 1600G 空间里

的一个地址下标。然后根据计算出来的 Hash 值将对应的地址空间里的比特值设为 1，这样一个 MD5 就可以将 8 个比特位设置为 1。

如果要检查一个 MD5 是否存在，只需要让 MD5 重复使用这 8 个哈希算法，计算出 8 个地址下标，然后检查它们里面的二进制数是否全是 1，如果是 ，那么表示这个 MD5 已经存在了。所以，在海量 MD5 中检查一个 MD5 是否存在，布隆过滤器会比哈希表更节约内存空间。

小结

因为数据存储是有状态的，所以海量数据存储的分布式架构要解决的核心问题就是：在一个有很多台服务器的分布式集群中，如何知道数据存储在哪台服务器上？

解决方案有两种，一种是有专门的服务器记录数据存储在哪里，即有一个**元数据服务器**。HDFS 里的 NameNode 和 HBase 里的 HMaster 都是这样的角色。应用程序想访问数据，需要先和元数据服务器通信，获取数据存储的位置，再去具体的数据存储服务器上访问数据。

另一种解决方案是通过某种算法计算要访问的数据的位置，这种算法被称作**数据路由算法**。分片数据库的硬编码算法就是一种数据路由算法，根据分片键计算该记录在哪台服务器上。MyCAT 其实也是采用路由算法，只不过将硬编码的分片逻辑记录在了配置文件中。

软件开发技术是一个快速发展的领域，各种新技术层出不穷，如果你只是被动地学习这些技术，很快就会迷失在各种技术细节里，疲惫不堪，最终放弃。事实上，每种技术的出现都因为要解决某个核心问题，最终诞生几种解决方案。同时，每种方案又会产生自己的新问题，比如分布式存储的数据的高可用，以及高可用带来的数据一致性，又需要产生相应的解决方案。

但是只要把握住核心问题和解决方案，就可以自己分析、推导各种衍生的问题和方案，思考各种优缺点和改进策略，最终理解、掌握一个新的技术门类。这不是通过辛苦学习，来掌握一个技术，而是从上帝视角，站在和这些技术的创造者一样的维度去思考，最终内化到自己的知识体系中。

思考题

分布式存储中有个非常著名的数据路由算法，叫一致性 Hash 算法，这个算法要解决的问题是什么？解决的思路是什么？

欢迎在评论区分享你的思考，我们共同进步。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

生成海报并分享

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 06 | 短视频系统设计：如何支持三千万用户同时在线看视频？

下一篇 08 | 秒杀系统设计：你的系统可以应对万人抢购盛况吗？

精选留言 (6)

写留言

阿卷
2022-03-02

普通的hash算法是通过数据与机器数取模运算来将数据映射到具体的节点，如果有机器加入和退出，则所有的数据都需要重新映射，这将会导致大量的数据迁移。
一致性hash就是用来解决这个问题的，一致性hash算法可以在加减节点时尽可能的保证现有的映射关系不变，从而减少数据迁移量。
大致的实现思路就是预先设置一个环形节点空间，比如环上预设1万个节点，将机器id通...
展开

作者回复: 非常赞

3

hao
2022-03-03

老师，布隆过滤器应该只能保证数据肯定不存在，并不能保证数据一定存在，就算八个比特位都匹配了，但是仍然有几率是其他数据占用的

作者回复: 是的，所以布隆过滤器的使用要看场景，能否接受误差，以及是否对误差有修正方案。

短URL中，如果布隆过滤器错误判定短URL重复，而实际并不重复，后果知识浪费几个短URL而已，完全可以接受。

网盘中，布隆过滤器判断重复后，还要到数据库中查找，不会造成错误。

Ronnie
2022-03-03

一致性哈希解决了哈希算法在增加服务节点时，需要更改数据路由算法逻辑的问题

作者回复: 赞

2022-03-03

一致性hash：
首先先说hash，就是为了把数据均匀分布到每一个节点上。根据数据的hash值，hash值再跟节点数了取余，就得出了节点编号。但是节点如果扩容，所有节点就需要重新将数据初始化。这是我们不愿意看到的。
…
展开

作者回复: 非常赞

peter
2022-03-02

请教老师一个问题：
Q1：HBase图例中，Zookeeper是独立于HMaster的集群吗？
如果是独立的集群，zookeeper不了解HMaster的情况，根据什么来选主？
或者，另外一种情况是：Zookeeper和HMaster运行在同样的机器上，一台HMaster所在的

机器上就有一个Zookeeper，这样zookeeper才能根据HMaster的情况选主；这种情况下，...
展开

作者回复: ZooKeeper是独立的，通常公司还有其他系统也要使用ZooKeeper，ZK是公司级共用的。
HMaster作为ZK的客户端需要和ZK通信，ZK是了解HMaster的，HMaster也要监听ZK的数据变化

Geek_7347cf
2022-03-02

什么是cap讲了 为什么会有cap好像没讲

作者回复: 嗯，这部分内容文中讲的不明确，但其实是不断提及的，CAP的核心问题是源于数据分布式存储的一致性，多备份、海量数据存储的一致性几乎是无解的。